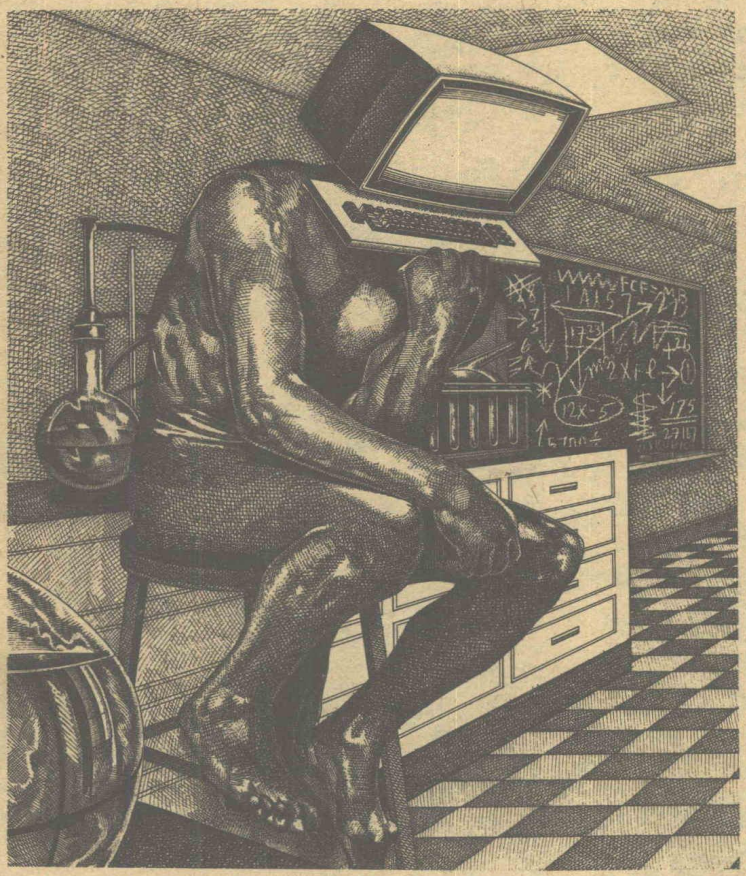
ปัญญาประดิษฐ์ในงานวิศวกรรมซอฟท์แวร์



ครรชิต มาลัยวงศ์*

"การประยุกต์ปัญญาประดิษฐ์เข้ากับงานวิศวกรรมซอฟท์แวร์นั้นเป็นเรื่องที่จำเป็นมาก ถ้าหากประยุกต์ไม่ได้ก็ไม่ควรมีงานปัญญาประดิษฐ์อีกต่อไป"

นั่นเป็นคำประกาศของนักซอฟท์แวร์ญี่ปุ่นคนหนึ่งในระหว่างที่ผมกับเขารับประทานอาหารร่วมกันเมื่อเร็ว ๆ นี้

ท่านผู้อ่านที่สนใจติดตามเรื่องที่ผมสนทนากับนักคอมพิวเตอร์ชาติต่าง ๆ ซึ่งผมได้นำมาเล่าสู่กันฟังเป็นประจำใน "ไมโครคอมพิวเตอร์" นี้ อาจจะจำได้ว่าผมเคยเชิญนักซอฟท์แวร์ญี่ปุ่นท่านหนึ่งมาสนทนาเรื่องวิศวกรรมซอฟท์แวร์มาแล้ว นักซอฟท์แวร์ท่านนี้ ชื่อนายมาซาโอะ มัตสุโมโต ซึ่งหลังจากที่ผมเชิญให้มาบรรยายที่เอไอทีแล้ว ต่อมาสมาคมคอมพิวเตอร์ก็เชิญท่านผู้นี้มาบรรยายพิเศษในงาน COMPUTER THAI'85 เมื่อเดือนมีนาคม ปีที่แล้ว

นายมัตสุโมโตส่งข่าวให้ผมทราบว่า ขณะนี้ทางห้องปฏิบัติการซอฟท์แวร์ของบริษัท NEC กำลังขมีขมันประยุกต์ปัญญาประดิษฐ์กับงานวิศวกรรมซอฟท์แวร์ ถ้าผมสนใจและมีเงินพอจะจ่ายค่าเครื่องบินให้แกได้ แกก็จะบินมาเล่าให้ฟัง ผมคิดดูแล้วเห็นว่าขืนปล่อยให้แกบินอยู่ฝ่ายเดียวก็จะเอาเปรียบแกมากเกินไป สู้ผมบินไปคุยกับแกเลยไม่ได้ ดังนั้นผมจึงควักกระเป๋าอันยอบแยบซื้อตั๋วเครื่องบินไปคุยกับ มร.มัตสุโมโตถึงโตเกียวเลยทีเดียว ทั้งนี้ทั้งนั้นก็เพื่อท่านผู้อ่านไมโครคอมพิวเตอร์แท้ ๆ เลยนะเนี่ย

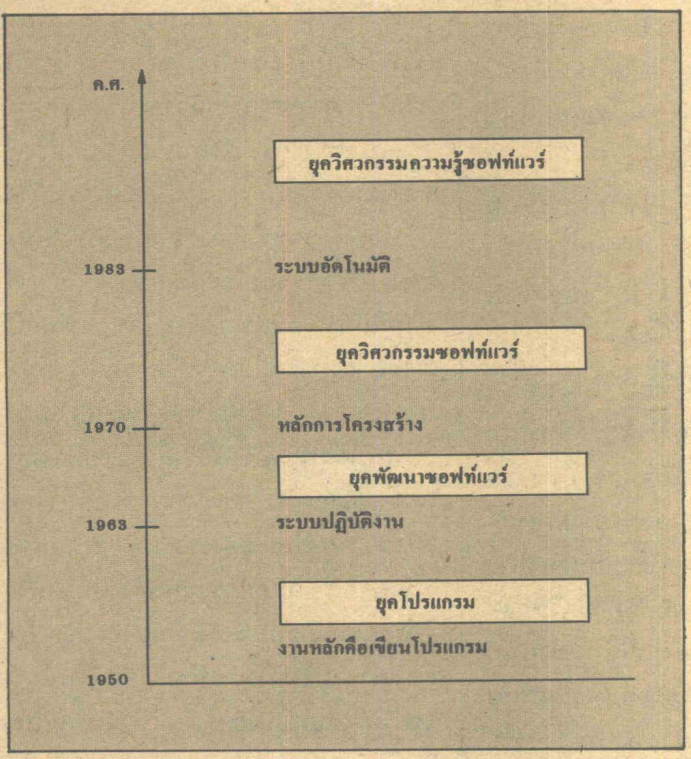
ผมมาถึงห้องปฏิบัติการซอฟท์แวร์ของบริษัท NEC ในเขตมินาโตกุ นครโตเกียวตรงตามนัด โดยมีมร.คูโบ มาซาฟูมิ ผู้ประสานงานฝ่ายเอเชียอาคเนย์เป็นผู้นำทางฝ่าการจราจรที่สับสนวุ่นวายผสมกับหลงทางอีกสองสามเลี้ยว

มร. มาซาโอะ มัตสุโมโต ยังคงแคล่วคล่องเหมือนเดิม แต่มีท่าทีอิดโรยบ้าง เพราะเขาเพิ่งกลับจากประชุมวิชาการคอมพิวเตอร์ที่ปักกิ่ง เมื่อสามวันก่อนนี้เอง เขาหัวเราะชอบใจเมื่อผมตั้งฉายาว่าเขาเป็น Travelling Researcher ไปแล้ว (ปัญหาทางด้านการเขียนโปรแกรมและวิจัยปฏิบัติการมีเรื่องหนึ่งเกี่ยวกับ Travelling salesman ซึ่งเป็นการหาเส้นทางที่ประหยัดที่สุดที่นักขายจะแวะเมืองต่าง ๆ ให้ได้ครบทุกเมือง)

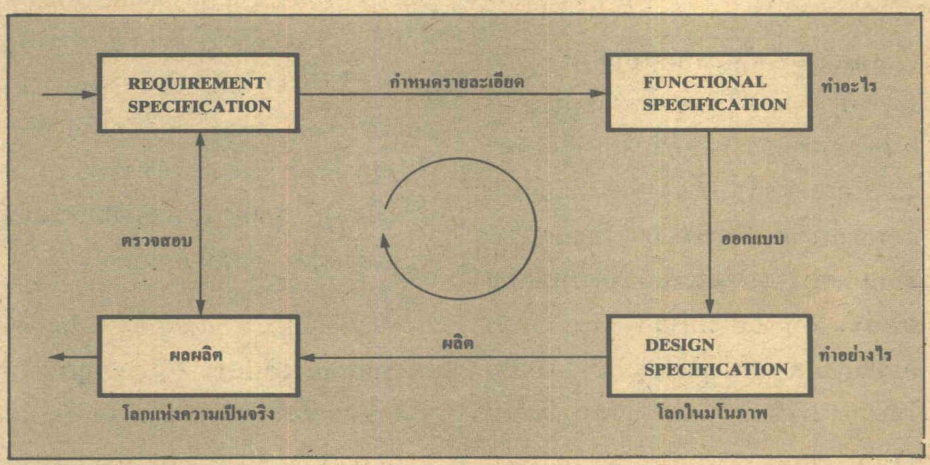
"เรามีเวลาน้อย ดังนั้นควรเข้าเรื่องของเราเลยดีไหม" มร.มัตสุโมโตพูดพร้อมกับหยิบแผ่นใสที่มีรายละเอียด ดังรูปที่ 1 ออกจากแฟ้มมาให้ผมดู "ผมอยากจะเริ่มด้วยประวัติแนวคิดของการพัฒนาซอฟท์แวร์ก่อน แรกเริ่มเดิมทีที่เราใช้คอมพิวเตอร์นั้น เราอยู่ในยุคของการเขียนโปรแกรมอย่างรูปนี้คือไม่ว่าคุณจะทำอะไร คุณต้องเขียนโปรแกรมทั้งนั้น

*อดีตคณบดี คณะคอมพิวเตอร์ประยุกต์ สถาบันเทคโนโลยีแห่งเอเชีย

และต้องเขียนเป็นภาษาเครื่องด้วย ต่อมาราวค.ศ. 1963 ก็เริ่มเกิดระบบ Operating System และภาษาระดับสูงขึ้น ทำให้สะดวกต่อการเขียนโปรแกรม และทำให้มีผู้สนใจใช้คอมพิวเตอร์มากขึ้นด้วย ช่วงนี้ได้เกิดความพยายามสร้างซอฟท์แวร์ขนาดใหญ่ ๆ ขึ้น ดังนั้นผมจึงเรียกว่าเป็นยุคพัฒนาซอฟท์แวร์ พอมาถึง ค.ศ. 1970 ได้เกิดความสนใจที่จะทำให้การเขียนโปรแกรมเป็นวิทยาศาสตร์มากขึ้น และเกิดหลักการด้านโปรแกรมโครงสร้างขึ้น ช่วงนี้ผมเรียกว่าเป็นยุควิศวกรรมซอฟท์แวร์ ในยุคนี้เองได้มีผู้พยายามทำให้การเขียนและพัฒนาซอฟท์แวร์ มีลักษณะอัตโนมัติมากขึ้น เช่น มีการทำโปรแกรมช่วยสร้างฟอร์แมทของจอภาพ โปรแกรมสำหรับสร้างโปรแกรมประยุกต์ต่าง ๆ แต่งานเหล่านี้ยังมีลักษณะกระจัดกระจายไม่ต่อเนื่อง จนมาถึงในราวปี ค.ศ. 1983 จึงมีผู้พยายามรวมระบบโปรแกรมอัตโนมัติเข้าด้วยกัน และทางห้องปฏิบัติการของผมเองก็เริ่มคิดนำปัญญาประดิษฐ์มาประยุกต์เข้าไปด้วย จึงเกิดเป็นยุค "วิศวกรรมความรู้ซอฟท์แวร์ (Software knowledge engineering)" ขึ้น



รูปที่ 1
ประวัติเทคโนโลยีการพัฒนาซอฟท์แวร์



รูปที่ 2 วัฏจักรการพัฒนาระบบงาน

"ทำไมคุณถึงคิดว่าจะนำเรื่อง ปัญญาประดิษฐ์มาใช้กับการเขียนโปรแกรมได้ล่ะครับ" ผมถามขัดจังหวะ

"ประการแรกก็คือ งานสร้างซอฟท์แวร์นั้น เป็นงานที่ต้องใช้ปัญญาความรู้ใช่ไหม ถ้าเราไม่มีความรู้เรื่องภาษาเขียนโปรแกรม เราก็เขียนโปรแกรมไม่ได้ ถ้าเราไม่รู้ลอจิกและวิธีการแก้ปัญหา เราก็สร้างโปรแกรมไม่ได้ แล้วนอกจากนั้นกระบวนการควบคุมขั้นตอนของการสร้างซอฟท์แวร์ก็ต้องใช้คนทำเป็นส่วนใหญ่ ดังนั้น "คน" จึงเป็นส่วนประกอบสำคัญของการสร้างซอฟท์แวร์ คราวนี้ลองดูเรื่องของปัญญาประดิษฐ์บ้าง เรื่องนี้ก็เป็นงานที่เกี่ยวข้องกับการใช้ปัญญาหรืออีกนัยหนึ่งเป็นการนำ "ปัญญา" เข้าไปบรรจุไว้ในเครื่องคอมพิวเตอร์ ดังนั้นจึงเป็นธรรมดาที่เราจะต้องคิดนำ "ปัญญา" ในการเขียนและสร้างซอฟท์แวร์เข้าไปบรรจุในเครื่อง ให้มันเขียนโปรแกรมได้เอง"

"นับเป็นความเห็นที่ดีครับ ว่าแต่ว่าเราจะบรรจุ"ปัญญา" เข้าไปอย่างไรเท่านั้น"

"อย่าเพิ่งรีบร้อนสิครับ การบรรจุ "ปัญญา" นั้นเป็นเพียงส่วนหนึ่งเท่านั้นของงาน เรายังมีเรื่องอื่นที่ต้องพิจารณาด้วยเหมือนกัน ลองดูรูปนี้สิครับ รูปที่ 2 นี้เป็นงานสำคัญสี่ส่วนของการสร้างซอฟท์แวร์ เราเริ่มจากการกำหนดความต้องการของผู้ใช้ซึ่งเรียกว่าเป็น Requirement Specification อันเป็นงานสำคัญอันดับแรกที่บอกให้เราทราบว่าผู้ใช้ต้องการอะไรบ้าง วิศวกรซอฟท์แวร์จะศึกษาข้อกำหนดนี้แล้วเขียน Functional Specification ซึ่งเป็นรายการที่บอกว่าซอฟท์แวร์จะต้องทำอะไรบ้าง หรือบางครั้งเราก็เรียกว่าเป็น Logical Design

นั่นเอง ผ่านจากขั้นนี้ไปก็เป็นการทำ Physical Design ซึ่งบอกว่าโปรแกรมหรือซอฟท์แวร์นั้นจะต้องทำงานอย่างไร แล้วที่สุดก็คือการเขียนโปรแกรมเพื่อให้ทำงานจริง ๆ ได้ ขอให้จำเรื่องนี้ให้ดีนะครับ ผมจะกลับมาพูดเรื่องนี้ใหม่อีกครั้ง"

"ดู ๆ แล้วก็เป็นเรื่องของวัฏจักรซอฟท์แวร์นั่นเอง" ผมให้ความเห็น

"ครับก็ทำนองนั้น เพียงแต่ผมไม่ได้เน้นที่ขั้นตอน หากเน้นที่ผลที่ได้จากการทำงานหลักสี่ประการ" มร.มัตสุโมโตตอบ "คราวนี้เรามาพิจารณาวิธีการพัฒนาซอฟท์แวร์ที่ใช้กันมาตั้งแต่เริ่มแรกจนถึงปัจจุบันนี้ วิธีแรกผมเรียกว่าวิธีพื้นฐาน ซึ่งใช้กันมาตั้งแต่ไหนแต่ไรแล้ว เป็นวิธีที่ไม่ได้ใช้เทคนิคหรือเครื่องมือใด ๆ เข้าช่วยเลย แล้วก็เดินตามขั้นตอนของวัฏจักรซอฟท์แวร์อย่างเคร่งครัดเลยทีเดียว"

"เป็นวิธีแบบ Tradition ใช่ไหมครับ รู้สึกผมจะเรียกแบบนั้น" ผมถาม

"คุณจะเรียกแบบนั้นก็ได้ไม่เป็นไร แนวคิดอยู่ที่ว่ายังไม่ได้ใช้เครื่องมือเข้าช่วยมากกว่า"

"แต่ก็ไม่ผิดใช่ไหมครับ เพราะขณะนี้ใคร ๆ ก็ยังใช้กันอยู่" ผมซัก

"ไม่ผิดครับ เป็นวิธีที่ใช้ได้ แต่อาจจะมีข้อจำกัดและไม่สะดวกหลายอย่าง ซึ่งผมจะอธิบายต่อไป" มร.มัตสุโมโตตอบ "วิธีการแบบที่สอง ผมเรียกว่าเป็นวิธีวิวัฒนาการ คือเป็นวิธีที่เริ่มจะนำเครื่องมือใหม่ ๆ มาใช้แล้ว เช่น ใช้ Data Flow Diagram หรือใช้เครื่องมือทดสอบซอฟท์แวร์เป็นต้น ส่วนเทคนิคใหม่ ๆ นั้นยังไม่ได้คิดนำมาใช้กันเท่าใดนัก วิธีนี้ปัจจุบันมีผู้ใช้กันมากขึ้น เพราะมีลักษณะการทำงานที่เป็นระบบดีขึ้น และวิธีที่สาม ซึ่งผมจะอธิบายต่อไปนั้นเป็นวิธีปฏิวัติ คือพลิกรูปโฉมของการพัฒนาซอฟท์แวร์ใหม่หมด โดยใช้ทั้งวิธีการแบบใหม่ และเครื่องมือแบบใหม่หมด"

"น่าสนใจครับ หวังว่าคงจะปฏิวัติสมชื่อ ไม่ใช่เพียงปฏิรูปเฉย ๆ" ผมเย้า

"ปฏิวัติจริง ๆ ครับ ขอให้ดูรูปนี้ก็แล้วกัน" มร.มัตสุโมโตหัวเราะพลางหยิบแผ่นใสขึ้นมาให้ดูอีกหนึ่งแผ่นดังรูปที่ 3 "รูปนี้ผมเปรียบเทียบการสร้างซอฟท์แวร์ที่ใช้อยู่ในปัจจุบัน กับวิธีปฏิวัติที่เราคิดขึ้น ขอให้พิจารณาวิธีที่เราใช้กันก่อนว่า เริ่มต้นด้วยจุดมุ่งหมายซึ่งอาจยังไม่ชัดเจนมากนัก แล้วก็สร้างเป็นข้อกำหนดความต้องการขึ้นว่า ผู้ใช้ต้องการให้ซอฟท์แวร์ทำอะไรได้บ้าง ที่จุดนี้คุณจะเห็นว่า ข้อกำหนดหรือสเปคนั้นมีลักษณะเป็นแบบ informal"

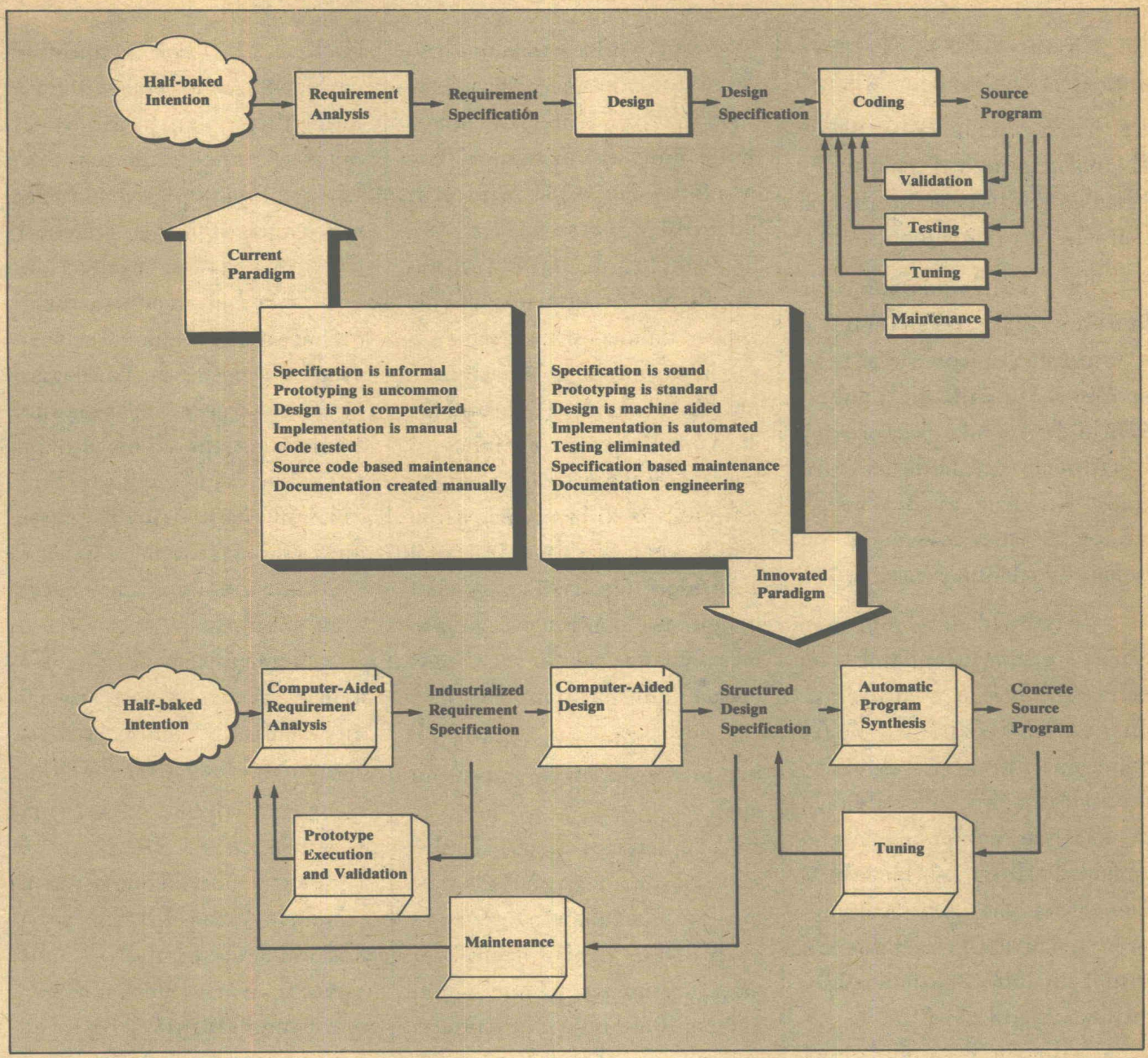
"แต่บางทีเราก็ใช้สเปคที่รัดกุมเหมือนกันนะครับ เช่น เขียนเป็นคำสั่งลำลอง แบบ Pseudo Code หรือ Sturctured English" ผมแย้ง

"นั่นเป็นสเปคที่ดีขึ้น" มร. มัตสุโมโตพยักหน้า "แต่ยังไม่ถึงขั้นที่ผมต้องการสเปคที่ผมต้องการเป็นพวก Executable specification ครับ เอาละครับ ถัดจากเรื่องของสเปค ก็มีเรื่องการทำต้นแบบ (Prototyping) ซึ่งวิธีที่ใช้กันอยู่ในปัจจุบันยังไม่ใคร่ใช้กันมากนัก การออกแบบก็ยังไม่ได้ใช้คอมพิวเตอร์เข้าช่วยยังคงนั่งคิดเขียนผังงานบ้าง หรือสร้างอัลกอริทึมบ้าง การเขียนโปรแกรมก็ต้องใช้นักโปรแกรมเขียนทีละโปรแกรม เขียนเสร็จแล้วก็ต้องทดสอบอีก แล้วปัญหาสำคัญยังมีอีกสองเรื่องคือการสร้างคู่มือซอฟท์แวร์ และการบำรุงรักษาโปรแกรมซึ่งล้วนต้องใช้คนทำทั้งนั้น นอกจากนั้นขอให้พิจารณาไดอะแกรมตอนบนของรูป คุณจะเห็นว่าการทดสอบซอฟท์แวร์ การปรับปรุง การบำรุงรักษาล้วนแล้วแต่ วนเวียนอยู่กับส่วนที่เป็นโปรแกรมที่เขียนขึ้นตลอดเวลา ดังนั้นงานสร้างและใช้ซอฟท์แวร์จึงมาหนักเอาที่ส่วนที่เป็นโปรแกรมหรือ source code นี่เอง"

"ถ้าอย่างนั้นวิธีปฏิวัติก็ต้องแก้ที่จุดนี้" ผมเอ่ยแทรกขึ้น

"ถูกทีเดียวครับ วิธีปฏิวัติของเราจะมีลักษณะเป็นการอัตโนมัติหมด อย่างที่เห็นในไดอะแกรมตอนล่าง เริ่มต้นด้วยคุณมีจุดมุ่งหมายที่ยังไม่ค่อยชัดเจนเหมือนเดิม แต่คราวนี้เราจะใช้เครื่องคอมพิวเตอร์เข้าช่วยในการวิเคราะห์ความต้องการของคุณ แล้วจะสร้าง Executable specification ออกมาให้เรา สเปคชุดนี้แหละครับที่เรานำไปใช้ในการทำต้นแบบสำหรับให้ผู้ใช้ซอฟท์แวร์ของเราตรวจสอบว่าผลของซอฟท์แวร์จะตรงกับความต้องการหรือไม่ ถ้าหากไม่ตรง หรือมีข้อผิดพลาดอื่น ๆ ก็จะมีการแก้ไขวิเคราะห์ความต้องการออกมาเป็นสเปคชุดใหม่ซึ่งตรงกับความต้องการของผู้ใช้มากขึ้น ๆ ที่สุดพอได้สเปคที่ลงตัวแล้วเราก็มาถึงการออกแบบซึ่งนำเอาสเปคนั้นมาแจกแจงว่าจะทำอย่างไรจึงจะเป็นโปรแกรมที่ใช้งานได้ ขั้นนี้เราก็ใช้คอมพิวเตอร์เข้าช่วยด้วยครับ ผลที่ได้จากขั้นนี้เราเรียกว่าเป็น Structured Design specification ซึ่งจะถูกส่งไปสังเคราะห์สร้างขึ้นมาเป็นโปรแกรมอีกต่อหนึ่ง แล้วคุณจะปรับปรุงการทำงานให้ดีขึ้นอย่างไรก็ได้ในขั้นนี้ ขอให้สังเกตว่าเมื่อคุณใช้วิธีใหม่นี้ คุณไม่ต้องทดสอบโปรแกรมอีกต่อไปแล้ว นอกจากนั้นการทำเอกสารคู่มือก็ไม่จำเป็นอีกแล้ว เพราะเราใช้วิธีทำเอกสารโดยอัตโนมัติ ซึ่งเราเรียกว่าเป็น Documentation engineering ข้อดีอีกอย่างก็คือต่อจากนี้ไปการบำรุงรักษาโปรแกรมก็ง่ายขึ้นมาก ทุกอย่างทำโดยคอมพิวเตอร์หมด เราไม่ต้องปวดหัวอีกแล้ว"

"ดีครับ... แต่ผมยังสงสัยเรื่องการทดสอบโปรแกรม ที่คุณบอกว่าไม่จำเป็นอีกแล้ว" ผมซักถามเรื่องซึ่งเป็นปัญหาใหญ่ของการเขียนโปรแกรมมาทุกยุคทุกสมัย "ถ้าเราไม่ทดสอบเรา



รูปที่ 3 เปรียบเทียบวิธีปัจจุบันกับวิธีใหม่

จะรู้ได้อย่างไรครับว่าโปรแกรมของเราจะทำงานถูกต้อง ไม่ได้ใส่สูตรหรือวิธีการทำงานที่ผิดเข้าไป"

"เรื่องทดสอบนี้ ผมใช้ศัพท์ต่างกันน่ะครับ" มร.มัตสุโมโตตอบ "การทดสอบที่ผมบอกว่าไม่จำเป็นอีกแล้วนั้นเป็นการตรวจว่าโปรแกรมผิดไวยากรณ์หรือเปล่า ซึ่งวิธีการใหม่ของเรานี้ย่อมจะมีผิดไม่ได้ เพราะเราสังเคราะห์โปรแกรมขึ้นโดยอัตโนมัติ ส่วนการทดสอบว่าตรงกับความต้องการของผู้ใช้หรือไม่นั้น เราใช้หลักการของการทำต้นแบบ (Prototyping) และการทดสอบว่าโปรแกรมถูกต้องหรือไม่เราใช้คำว่า validation"

"ถ้าอย่างนั้น ผมก็ยอมรับฟังได้ละครับ แต่ผมยังมีข้อข้องใจอีกหลายเรื่องเกี่ยวกับวิธีการสร้างซอฟท์แวร์แบบปฏิวัติตามที่อธิบายมานี้"

"ถามได้เลยครับ อย่าเก็บข้อข้องใจกลับไปถึงกรุงเทพฯโน่น" มร.มัตสุโมโตกล่าวอย่างเชื่อมั่นในตนเองมาก

"ข้อแรกก็คือเรื่องของ Executable Specification คุณใช้วิธีไหนครับ"

"พอถามก็จะเล่นเอาผมตอบไม่ได้เชียวนะ เรื่องนี้ผมไม่ขอตอบตรง ๆ เพราะเราใช้หลายแบบ คุณอาจจะใช้วิธี PSL/PSA หรือ Problem Specification Language และ Analyzer หรือพวก Specification Description Language ก็ได้"

"คำถามต่อไปก็คือ วิธีของคุณที่ทำมาแล้วนี้จะใช้กับงานประเภทใดได้บ้าง"

"ใช้ได้ทั้งนั้นแหละครับ ไม่ว่าจะเป็นงานด้านวิทยาศาสตร์ งานด้านธุรกิจ หรือแม้แต่งานด้าน Operation System เราก็ทำได้" มร.มัตสุโมโตตอบ

"ฟังดูไม่น่าเชื่อ แต่อาจเป็นเพราะผมยังไม่ทราบรายละเอียดชัดเจนก็ได้ คราวนี้ผมขอถามปัญหาสำคัญนะครับ คือเท่าที่คุณเล่ามาทั้งหมดนี้ ผมเห็นว่ายังไม่ได้ใช้หลักการด้านปัญญาประดิษฐ์ที่ตรงไหนเลย อาจเป็นเพียงแค่ Integrated technique เท่านั้น คือนำเอาเทคนิคหลาย ๆ อย่างมาใช้ร่วมกัน" ผมยิงปัญหาที่ข้องใจเป็นอย่างมากออกไป

"อาจเป็นได้ว่าผมยังอธิบายไม่ละเอียด" มร.มัตสุโมโตพยักหน้า "ผมอยากจะชี้ให้เห็นว่า การเขียนโปรแกรมเป็นงานแก้ปัญหาที่ซับซ้อน ต้องใช้กฎเกณฑ์ทางภาษาคอมพิวเตอร์ซึ่งมีอยู่มากมายเข้าประกอบ และต้องใช้ลอจิกที่ถูกต้อง การจะทำให้คอมพิวเตอร์ออกแบบและเขียนโปรแกรมได้โดยอัตโนมัติ จึงต้องสร้างฐานความรู้สำหรับเก็บกฎเกณฑ์ต่าง ๆ ทางภาษา และการสังเคราะห์โปรแกรมเอาไว้ใช้ ส่วนนี้แหละที่เป็นหน้าที่ของปัญญาประดิษฐ์"

"ฐานความรู้ที่จัดทำขึ้นนั้นใช้วิธีแบบไหนครับ"

"เราใช้หลายวิธีมีทั้งที่เป็น Frame-based, Rule-based และ Semantic Network เราใช้ผสมผสานกันหมด"

"วิธีที่คุณมัตสุโมโตเล่าให้ผมฟังนี่มีใช้หรือยังครับ"

"เราได้ทำสำเร็จเป็นบางส่วนแล้วครับ ให้ชื่อว่า SEA/I ซึ่งย่อมาจาก Software Engineering Architecture /I ทางที่ดีคือเราไปชม Demo กันในห้องปฏิบัติการของผมเลยจะได้เห็นชัดเจนกว่านั่งคุยกันอย่างนี้"

มร.มัตสุโมโต พาผมขึ้นลิฟต์ไปชั้นสี่ซึ่งส่วนหนึ่งเป็นห้องปฏิบัติการวิศวกรรมซอฟท์แวร์ ท่านผู้อ่านบางคนอาจหลับตานึกภาพเห็นห้องทดลองอิเล็กทรอนิกส์ มีเครื่องไม้เครื่องมือเกะกะ มีสายไฟฟ้าระโยงระยางอะไรอย่างนั้น แต่เปล่าครับ ห้องปฏิบัติการของเขาเป็นเพียงห้องทำงานขนาดใหญ่ตั้งโต๊ะเรียงต่อกันเป็นแถว ๆ และมีฉากเตี้ย ๆ กั้นไว้เป็นแนวเท่านั้น บนโต๊ะมีแต่หนังสือและคู่มือวางเกลื่อน และที่ขาดไม่ได้ก็คือไมโครคอมพิวเตอร์และเทอร์มินัล ที่ประจำอยู่โต๊ะละตัวหรือสองตัว แถมบางส่วนยังเป็นที่ตั้งไมโครคอมพิวเตอร์เป็นกลุ่ม ๆ ด้วย เท่าที่ผ่านตารู้สึกว่าจะมีไมโครคอมพิวเตอร์สารพัดเครื่อง เช่น แมคอินทอช ไอบีเอ็ม แวกซ์ แถมด้วยเครื่องมินิแวกซ์ 780 และของ NEC เองแยกเป็นสองห้องต่างหากด้วย

"ที่นี่มีนักซอฟท์แวร์กี่คนครับ" ผมถามเพราะเห็นเจ้าหน้าที่ของเขานั่งกันแออัด

"มีประมาณ 75 คน แล้วมีเจ้าหน้าที่พวก Support Staff อีกร้อยกว่าคน คนของเราไม่ใช่เป็นกลุ่มเดียวที่ทำซอฟท์แวร์ให้ NEC นะครับ ยังมีบริษัทในเครือ และกลุ่มย่อย ๆ อีกหลายกลุ่มที่รับผิดชอบด้านซอฟท์แวร์ ทางด้านเรานั้นทำงานวิจัยทางด้านซอฟท์แวร์เท่านั้น"

"แล้วที่ทำเสร็จจนถึงกับเป็นผลิตภัณฑ์ซอฟท์แวร์ออกสู่ตลาด มีบ้างไหมครับ"

"มีสิครับ อย่างเช่น SEA/I ที่จะแสดงให้ดูนี่อย่างไรล่ะครับ"

มร.มัตสุโมโต ขอให้นักซอฟท์แวร์ของเขาสองคนมาอธิบายและแสดงการใช้ SEA/I ให้ผมดู โดยมีสาวญี่ปุ่นคนหนึ่งคอยป้อนข้อมูลตามคำสั่ง

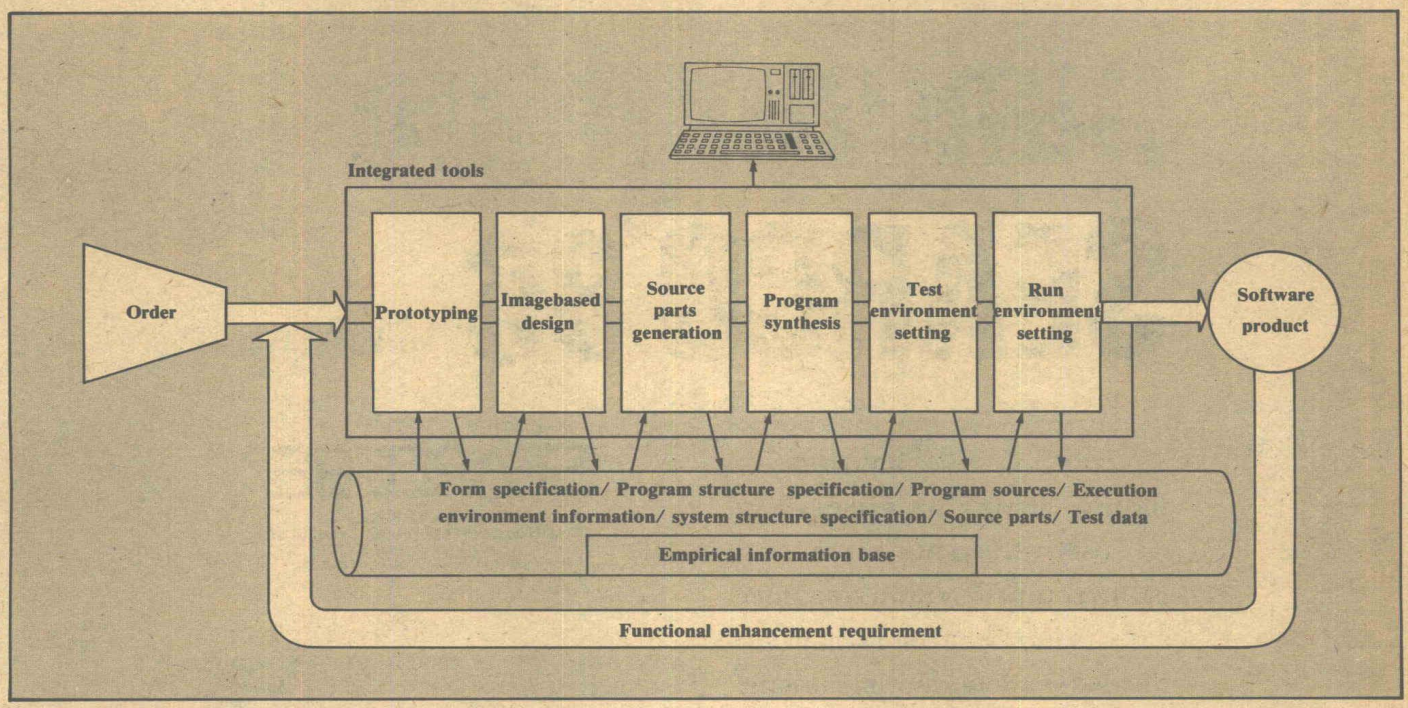
นักซอฟท์แวร์ลูกน้องมร.มัตสุโมโตนั้น พอจะพูดภาษาอังกฤษได้ เขานำเอาเอกสารเกี่ยวกับ SEA/I มาแจกพร้อมกับกล่าวว่า "รูปนี้ (รูปที่ 4) เป็นโครงสร้างของ SEA/I ในทัศนะของผู้ใช้ครับ ส่วนที่เป็นฐานความรู้ซึ่งเก็บความรู้เกี่ยวกับการออกแบบและบำรุงรักษาโปรแกรมนั้น แสดงอยู่ทางตอนล่างของภาพ ส่วนทางตอนบนเป็นเครื่องมือต่าง ๆ สำหรับใช้ในการพัฒนาซอฟท์แวร์เริ่มจากเครื่องมือช่วยในการกำหนดความต้องการ เครื่องมือช่วยทำต้นแบบ เครื่องมือช่วยสร้างโปรแกรม เครื่องมือสังเคราะห์โปรแกรม จนถึงเครื่องมือช่วยจัดสถานะการทำงานของซอฟท์แวร์ เครื่องมือเหล่านี้จะเรียกใช้ความรู้ที่เก็บอยู่ในฐานความรู้สำหรับนำมาช่วยในการทำงาน นอกจากนั้นพอซอฟท์แวร์ทำงานแล้วก็จะมีการประเมินผลนำข้อดีข้อเสียมาปรับปรุงระบบให้ดียิ่งขึ้นไปอีก"

"ผมพอเข้าใจแล้ว คราวนี้ผมอยากทราบว่า โปรแกรม SEA/I ของคุณนี้ทำอะไรได้บ้างครับ" ผมถามเพราะอยากเห็นการทำงานของ SEA/I สักที

"เราแบ่งงานของ SEA/I เป็นสองส่วนคือ งานวิเคราะห์ และสังเคราะห์ครับ งานส่วนวิเคราะห์ก็เป็นการรับทราบความต้องการของผู้ใช้ว่าต้องการออกรายงานที่มีลักษณะอย่างใดบ้าง อย่างเช่นขณะนี้ เราสามารถกำหนดลักษณะของรายงานบนจอภาพได้" เขาพูดพร้อมกับให้สัญญาณสาวญี่ปุ่นให้จัดการเรียกเมนูมาสร้างภาพบนรายงานบนจอ"

"ลักษณะส่วนนี้ก็คงจะคล้ายกับการออกแบบจอภาพนั่นเอง" ผมเปรยขึ้น

"ครับ แต่เราทำได้มากกว่านั้นหลายอย่าง เช่น เราเปลี่ยนแปลงฟอร์แมท ย้ายคอลัมน์ทีละมาก ๆ และกำหนดชื่อฟอร์มเอาไว้เรียกใช้ได้ ถ้าหากไม่กำหนดฟอร์มเราก็อาจกำหนดลักษณะของซอฟท์แวร์ได้ โดยใช้เมนูอีกส่วนหนึ่ง ซึ่งจะขอให้เราระบุรายละเอียดต่าง ๆ ของโปรแกรมหรือซอฟท์แวร์ให้ จากฟอร์มดีไซน์หรือการกำหนดรายละเอียดโปร-



รูปที่ 4 โครงสร้างของ SEA/I ในทัศนะผู้ใช้

แกรมนี้ SEA/I จะเริ่มสร้างลอจิกของโปรแกรมขึ้นโดยใช้เทคนิคที่เรียกว่า Program Tree ซึ่งมีลักษณะเป็นไดอะ-แกรมคล้าย ๆ กับแบบแนสซี ชไนเดอร์มาน และสุดท้ายของขั้นวิเคราะห์นี้ก็เป็นการสร้างคำสั่งขึ้นเป็นภาษาคอม-พิวเตอร์"

ระหว่างที่เขาอธิบายนั้น เขาก็สั่งให้ผู้ช่วยสาวกดแป้นพิมพ์ไปด้วยเป็นระยะ ๆ จนสุดท้ายหลังจากรออยู่ครู่ใหญ่ก็ได้เป็นโปรแกรมภาษาโคบอลออกมา

"งานขั้นต่อมาเป็นงานสังเคราะห์ ซึ่งระบบของเราจะนำโปรแกรมที่สร้างขึ้นมานั้น มาสังเคราะห์ จัดรวมกันเข้าเป็นโปรแกรมหรือระบบเดียวกัน" นายนักซอฟท์แวร์อธิบาย

"ผมเข้าใจว่าการสร้างโปรแกรมบางอย่างคุณน่าจะใช้หลักการของ sub-routine library ได้" ผมให้ความเห็น

"ใช่ครับ เราใช้หลักการแบบเดียวกัน แต่เราเรียกใหม่ว่าเป็น Reusability เพราะเราไม่ได้ทำในระดับ source code ที่จะเรียกใช้แต่โปรแกรมย่อยเท่านั้น เรายังใช้หลักการนี้ในระดับการออกแบบด้วย คือเราจะเก็บ specification ของแบบและโปรแกรมต่าง ๆ ที่เราเห็นว่าทำงานถูกต้อง และมีประสิทธิภาพเอาไว้ เมื่อ SEA/I ได้รับทราบกำหนดความต้องการของผู้ใช้แล้ว มันก็จะวิเคราะห์ว่ามีส่วนใดคล้ายคลึงกับแบบหรือโปร-แกรมที่เก็บเอาไว้แล้วบ้าง ถ้ามีมันก็หยิบส่วนนั้นออกมาเลย ไม่ต้องเสียเวลาสร้างโปรแกรมส่วนนั้นใหม่ ความจริงส่วนที่เป็นการออกแบบเอาท์พุทหรือฟอร์มนั้น ก็มีลักษณะเป็นแบบ Reusability นั่นแหละครับ" คราวนี้มร.มัตสุ-โมโต เฉลยปัญหาเอง

เย็นนั้นมร.มัตสุโมโตเชิญผมข้ามถนนไปรับประทานอาหารที่ร้านอาหารเล็ก ๆ แห่งหนึ่งตรงข้ามกับสำนักงานใหญ่ของบริษัท NEC เอง อาหารก็ไม่แคล้วปลาดิบที่ขึ้นชื่อลือชาของญี่ปุ่นไปได้ เราได้พูดคุยกันอีกมากมายหลายเรื่อง แต่ลงท้ายผมถามเขาว่า "คุณมัตสุ-โมโต ผมอยากถามคุณสักหน่อยว่า ถ้าคุณคิดสร้างระบบ SEA/I ของคุณสำเร็จหมดแล้ว ลักษณะการพัฒนาซอฟท์แวร์ของคนใช้คอมพิวเตอร์ก็จะเปลี่ยนแปลงไป แล้วถึงตอนนั้นน่ะ คุณจะให้สอนเนื้อหาอะไรในวิชาวิศวกรรมซอฟท์แวร์ล่ะครับ อย่างนี้ผมมิตกงานดอกหรือ"

มร.มัตสุโมโตหัวเราะ "คำถามที่ดีครับ ผมคิดว่าเนื้อหาของวิชาคงต้องเปลี่ยนไปแน่ ๆ แต่เรื่องวิศวกรรมซอฟท์-แวร์ยังไม่หมดเรื่องที่จะเขียน ที่จะทำวิจัยดอกครับ เราคงต้องหันไปเน้นที่การสังเคราะห์ หรือการเปลี่ยนสเปคให้เป็นโปรแกรม อะไรทำนองนี้"

ครับ... นั่นก็เป็นเพียงการคาดคะเนเท่านั้น ยังบอกอะไรไม่ได้ เพราะเทคโน-โลยีในวงการคอมพิวเตอร์เปลี่ยนเร็วเหลือเกิน วันหนึ่งเขียนโปรแกรมอยู่เพลิน ๆ อีกวันเขามาบอกว่า คุณ ๆ เขาเลิกเขียนโปรแกรมกันแล้ว และก็ไม่แน่ว่าอีกวันหนึ่งอาจจะมีคนวิ่งมาบอกว่า คอม-พิวเตอร์มีปัญญามากจนปฏิวัติเสร็จไปแล้วก็ได้